A Framework for Integrating Network Information into Distributed Iterative Solution of Sparse Linear Systems [⋆]

Devdatta Kulkarni and Masha Sosonkina

University of Minnesota, Duluth, MN 55812 USA
{kulk0015, masha}@d.umn.edu

Abstract. Recently, we have proposed a design of an easy-to-use network information discovery tool that can interface with a distributed application non-intrusively and without incurring much overhead. The application is notified of the network changes in a timely manner and may react to the changes by invoking the adaptation mechanisms encapsulated in notification handlers. Here we describe possible adaptations of a commonly used scientific computing kernel, distributed sparse large-scale linear system solution code.

Keywords: balancing computation communication, network information collection

1 Introduction

Distributing computation and communication resources is already a well-established way of solving large-scale scientific computing tasks. Communication library standards, such as Message Passing Interface (MPI) [13], make applications portable across various distributed computing platforms. However, for high-performance applications in distributed environments, the efficiency and robustness are difficult to attain due to the varying distributed resource – such as communication throughput and latency – availability at any given time. This problem is especially acute in computational grids, in which the resource pool itself may vary during the application execution. One way to handle this situation is to incorporate into application execution run-time adaptive mechanisms, which may complement the static (compile-time) adjustments. At present, however, there is a lack of easy-to-use tools for the application to learn the network information and to request particular network resources dynamically. It is desirable to have a mechanism that provides the network information transparently to the application programmer or user, so that the burden of handling the low level network information is shifted to a network developer. With a knowledge of network performance, the application may adapt itself to perform the communication more efficiently. The adaptation features are, of course, application-specific. For a scientific application, it may be beneficial to perform more local computations (iterations) waiting for the peer processors. This paper presents a general way to incorporate the network information and adaptation procedures into an application, that can be used by a wide range of scientific applications. Section 3 provides an overview of our design and implementation of the Network Information Collection and Application Notification (NICAN) tool [15]. It will be used to notify the chosen application of the changes in network status. In Section 4, we outline the distributed sparse linear system solution code used in the experiments. Specifically, the application under consideration is Parallel Algebraic Multilevel Solver (pARMS) [5], developed at the University of Minnesota and shown to be effective for solving large sparse linear systems. Section 5 describes an adaptation scenario for pARMS invoked in response to certain network conditions. We summarize the work in Section 7.

[⋆] This work was supported in part by NSF under grants NSF/ACI-0000443 and NSF/INT-0003274, and in part by the Minnesota Supercomputing Institute

2 Related Work

Providing network information to distributed applications has been widely recognized as an important task. Remos [6] makes available the runtime network information for the applications through an interface. A network independent API is provided which enables applications to get network topology information and per flow information. Netlogger [3] gives a detail logging of all the events occurring during the execution of a distributed application. Authors of Congestion Manager (CM) [1] show an effective way to adapt network applications based on a kernel module which enables similar flows, between same source and destination, to share the network congestion information. It allows applications to adapt by providing them with an API for querying the network status. While CM tries to capture the adaptability concept within the kernel bounds, it puts the responsibility of finding out the relevant network information upon the application. HARNESS [2] is a distributed virtual machine system having the capability of supporting plug-ins for tasks like communication, process control and resource monitoring [2]. Network Weather Service (NWS) [17], monitors network behavior and shows that this information can be successfully used for scheduling in distributed computational grid environments.

Our approach is different from these systems mainly in the scope of seamless interaction with application at runtime. We concentrate on specifically providing the adaptive capabilities to scientific distributed applications. Our results are rather application specific but the framework is general enough to be used with scientific applications having alternating computation communication characteristics. In [7], authors show dynamic adaptation of Jacobi-Davidson eigen solver based on the memory thrashing competing applications and CPU based loads. Their approach is similar to our approach but they do not yet address the network interface related issues. Decoupling of network information collection from the application execution and providing a mechanism to encapsulate application adaptations are the main features of NICAN. The developed tool is light weight to complement high computation and memory demands of large-scale scientific application.

3 Providing dynamic network information to applications

A major design goal is to augment the application execution with the knowledge of the network while requiring minimum modifications of the application and without involving the user/programmer into the network development effort. Indeed, the network information collector should have a negligible overhead and not compete with application for resource usage. This design requirement is especially vital since we target high-performance distributed applications which often demand full capacity of computer resources.

The NICAN accepts the request from the application and delivers the obtained network characteristics to the application. This enables supplying an application with the network information only *if this information becomes critical*, i.e., when the values for the network characteristics to be observed fall outside of some feasible bounds. The feasibility is determined by an application and may be conveyed to the network information collector as parameter. This *selective notification* approach is rather advantageous both when there is little change in the dynamic network characteristics and when the performance is very changeable. In the former case, there is no overhead associated with processing unnecessary information. In the latter, the knowledge of the network may be more accurate since it is obtained more frequently. Multiple probes of the network are recorded to estimate the network performance over a longer period of time. They may also be useful for the prediction of network performance in such common cases as when an iterative process lies at the core of application.

In NICAN, the process of collecting the network information is separated from its other functions, such as notification, and is encapsulated into a module that can be chosen depending on the types of the network, network software configuration, and the information to be collected. For example, assume that the current throughput is requested by an application during its execution. Then, if the network has the Simple Network Management Protocol (SNMP) [8] installed, NICAN will choose to utilize the SNMP information for throughput calculation. Otherwise, some benchmarking procedure – more general than probing SNMP but also more costly – could be applied to determine the throughput. To determine the latency between two hosts, the system utilities such as `ping` and `traceroute` can be used. NICAN collects latency independently of throughput. Whenever a network parameter value becomes available, NICAN processes it immediately without waiting for the availability of the other parameter values. Delaying the processing would cause the excessive overhead for NICAN and would lead NICAN to notify an application with possibly obsolete or wrong data. Figure 1 shows a modular design of NICAN/application interface. The NICAN implementation consists of two parts, the NICAN front end and the NICAN back end. NICAN front end provides for the application adaptation and the NICAN back end performs the network data collection. The application starts the NICAN back end by invocation of the NICAN back end thread. NICAN back end thread consists of separate threads for collecting different network parameters. Note that for simplicity NICAN does not attempt to perform a combined parameter analysis: each network parameter is monitored and analyzed separately from others. The modular design, shown in Figure 1, enables an easy augmentation of the collection process with new options, which ensures its applicability to a variety of network interconnections.

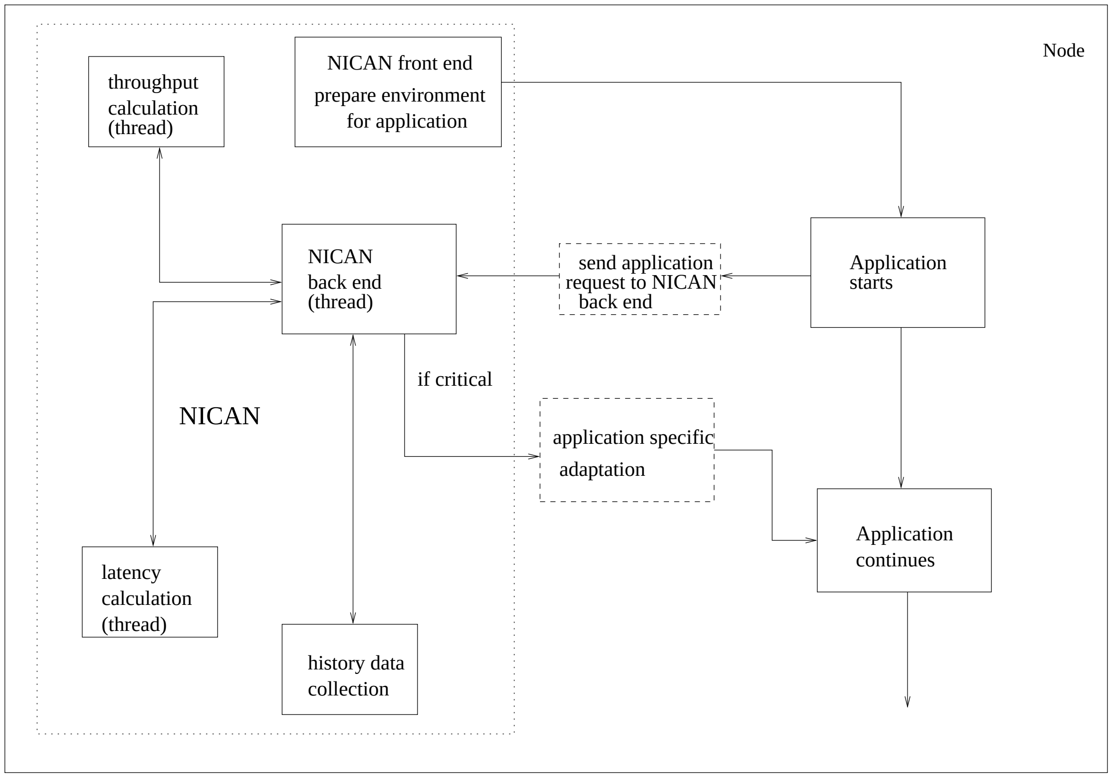
Figure 1 shows that the application starts the NICAN back end thread and passes the monitoring request to the NICAN informs the application about the changes in the network conditions in a timely fashion such that there is no instrumenting of an application with, say, call-queries directed to the network interface. In fact, the initialization of the NICAN tool may be the *only* non-application specific modification required in the application code to interface with NICAN. Upon the notification from the NICAN the application may need to engage its adaptive mechanisms. To minimize changes inside the application code, we propose to encapsulate application adaptation in a notification handler invoked when NICAN informs the application about the changes in the network conditions. This handler can contain an adaptation code with a possible access to some application variables. NICAN front end also provides for preparing the execution environment for better application execution by dynamically selecting relatively less loaded nodes from amongst the pool of nodes. A more detailed description of NICAN design and implementation can be found in [4].

For a distributed application that uses Message Passing Interface (MPI), the communication overhead also includes the overhead for MPI. Since most of the high performance computing applications use MPI to ensure portability across distributed environments, measuring and monitoring the MPI overheard may be useful for performance tuning. NICAN provides a way to interact with MPI-based distributed applications.

4 Distributed sparse linear system solution

Among the techniques for solving large sparse linear systems is a recently developed Parallel Algebraic Recursive Multilevel Solver (pARMS) [5]. This is a distributed-memory *iterative method* (see, e.g., [9] for the description of modern iterative methods) that adopts the general framework of distributed sparse matrices and relies on solving the resulting distributed Schur complement systems [10]. pARMS focuses on novel linear system transformation techniques, called *preconditioning* [9], which aim to make the system easier to solve by iterative methods. In particular, pARMS combines a set of domain decomposition techniques [12], frequently used in parallel computing environments, with multi-level preconditioning [16], which leads to a scalable convergence process with increase in problem size.

An iterative solution method can be easily implemented in parallel, yielding a high degree of parallelism. Consider, for example, a parallel implementation of FGMRES [5], a variation of a popular solution method, restarted Generalized Minimum RESidual algorithm (GMRES) [9]. If the classical Gram-Schmidt procedure is used in its orthogonalization phase, an iteration of the parallel algorithm has only two synchronization points, in which all-to-all processor communications are incurred.



Node
throughput calculation (thread)
NICAN front end prepare environment for application
NICAN back end (thread)
send application request to NICAN back end
Application starts
if critical
NICAN
application specific adaptation
Application continues
latency calculation (thread)
history data collection

Fig. 1. Overview of NICAN design

One way to partition the linear system $Ax = b$ is to assign certain equations and corresponding unknowns to each processor. For a graph representation of sparse matrix, graph partitioner may be used to select particular subsets of equation-unknown pairs (sub-problems) to minimize the amount of communication and to produce sub-problems of almost equal size. It is common to distinguish three types of unknowns: (1) Interior unknowns that are coupled only with local equations; (2) Inter-domain interface unknowns that are coupled with both non-local (external) and local equations; and (3) External interface unknowns that belong to other sub-problems and are coupled with local equations. Thus each local vector of unknowns x_i is reordered such that its sub-vector u_i of internal components is followed by the sub-vector y_i of local interface components. The right-hand side b_i is conformly split into the sub-vectors f_i and g_i , i.e.,

$$x_i = \begin{pmatrix} u_i \\ y_i \end{pmatrix} \ ; \quad b_i = \begin{pmatrix} f_i \\ g_i \end{pmatrix} \ .$$

When block-partitioned according to this splitting, the local matrix A_i residing in processor i has the form

$$A_i = \left(\begin{array}{c|c} B_i & F_i \\ \hline E_i & C_i \end{array}\right) ,$$

so the local equations can be written as follows:

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix} .$$

Here, N_i is the set of indices for sub-problems that are neighbors to the sub-problem i. The term $E_{ij}y_j$ reflects the contribution to the local equation from the neighboring sub-problem j. The result of the multiplication with external interface components affects only the local interface unknowns, which is indicated by zero in the top part of the second term of the left-hand side.

5 Integration of network information into pARMS

The iterative nature of the pARMS execution process is a typical example of most distributed iterative linear system solutions. In each iteration, local computations are alternated with the data exchange phase among all neighboring processors following the pattern of sparse matrix-vector multiplication. This pattern is preserved if a domain decomposition type preconditioner (see e.g., [12]) is used. For such a preconditioner, it is possible to change the amount of local computations in each processor depending on local sub-problem or computing platform characteristics. For varying sub-problem complexity, this issue has been considered in [11] and extended to encompass unequal processor loads in [14]. It has been shown that performing more local iterations in the less loaded processors and thus computing a more accurate local solution would eventually be beneficial for the overall performance. In other words, the accuracy would eventually propagate to other processors, resulting in a reduction of the number of iterations to converge. Here we describe how, with the information provided by NICAN, these adaptations can be carried out based on the changing network conditions.

pARMS uses MPI for communication between participating processors. The rendezvous of all the peers might not be at the same time since each processor might have a different computational load or incur delays in sending or receiving data on its network interface resulting in *low network interface throughput.* Thus, the knowledge of how busy the processor is and how much network interface throughput is available for the processor communications can help in devising adaptation strategies for pARMS.

5.1 Adaptation of pARMS based on network condition

If the local computations are balanced, then each processor completes its local computations and then waits, at a communication rendezvous point, for others to complete. Consider what happens after the first exchange of data. The processor which has more network interface throughput available for communication would complete its data transfer earlier relative to the other processors. Therefore this processor will start (and finish) its local computations early and incur an idle time waiting for the other processors. For a distributed iterative linear system solution performed on two processors, Figures 2 and 3 depict, respectively, the ideal scenario of balanced computations and communications and a scenario in which Processor 1 has a low network interface throughput. Instead of idling, Processor 2 can perform more local computations to obtain a more accurate solution and arrive at the rendezvous point later.

pARMS source code is instrumented with an initialization call for NICAN after the initialization of MPI. Within this function call the parameters to be monitored are passed. We have used network interface throughput as the parameter, and the notification criterion is set to reporting global maximum *achieved* throughput and local *achieved* throughput only when their values differ *substantially.* Note that a large, relative to the link nominal bandwidth, value for the achieved throughput would indicate reaching the capacity of the network interface and would lead to communication delays.

After NICAN initialization, pARMS continues its normal operation. NICAN, on the other hand, starts monitoring the achieved throughput for each processor using the SNMP protocol. Then the maximum achieved throughput is computed by NICAN among all the participating processors without interfering with pARMS communication or computation tasks. Upon meeting the notification condition, the values obtained for the global maximum and local throughput are signaled to the local application process and passed to the respective notification handler.

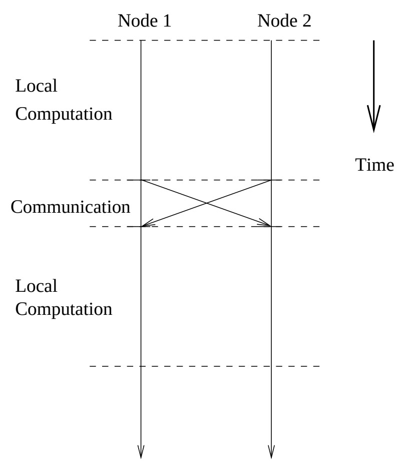


Fig. 2. Ideal scenario: Balanced computations and communications

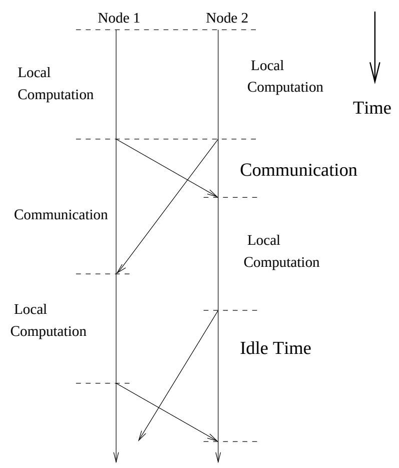


Fig. 3. A more realistic scenario: Balanced computations and unbalanced communications

The user may incorporate a desired adaptation strategy into the notification handler, thus avoiding direct changes to the application source code. If certain application variables need to be adjusted in response to the network performance, they can be shared with the handler, which contains the adaptation code. For example, when the pARMS adaptation consists of adjusting the number of local iterations n^i in the preconditioner application on processor i, the variable n^i is shared with the handler local to processor i.

Algorithm 5.11 outlines a procedure of incorporating adaptive features in pARMS using NICAN. For both sets of experiments, we construct a rather simple notification handler which, upon invocation, changes the value of a shared variable n^{inner} depending on which experiment is performed.

Algorithm **5.11** *Incorporating adaptive features in pARMS using NICAN*

```
in pARMS:
1.  Declare $n^{inner} = n^i$ as shared between pARMS and NICAN.
2.  Start pARMS outer iteration
3.      Do pARMS $n^{inner}$ inner iterations;
4.      Exchange interface variables among neighbors;
5.  End pARMS outer iteration.


in NICAN:
1.  Start handler when application is notified
2.      If (adaptation condition for experiment set One)
3.          $n^{inner} = n^{inner}$ + constant;
4.      If (adaptation condition for experiment set Two)
5.          $n^{inner} = n^{inner}$ + variable;
6.  End handler.
```

The time ${T_c}^i$ each node i spends in a communication phase may be determined knowing the current achieved throughput τ^i and the amount of data D^i to be communicated. The throughput for each node is calculated using SNMP. The amount of data D^i to be communicated is readily available from one of the pARMS data structures in each node. (The value of D^i is made accessible via the adaptation handler as well.) Specifically, ${T_c}^i$ is computed as follows:

$$ {T_c}^i = \frac{D^i}{B^i - \tau^i}, $$

where B^i is the link nominal bandwidth, the difference of which with τ^i gives the throughput available for the communication of data D^i. This formula will give us the time required for communicating D^i data values for the node. Among the neighbors the node having the largest value for this time will have the maximum usage of the network interface reflecting the competing network process running on that node. The maximum communication time $T_{\max}$ over all the nodes is calculated. The adjustment of the local iteration number is similar to the strategy proposed in [11]. In the (next) jth outer iteration of the iterative process

$$ n^i_j = n^i_{j-1} + \Delta^i_j, $$

where Δ^i_j, the number of iterations that node i can fit into the time to be wasted in idling otherwise at the jth outer iteration, is determined as follows:

$$ \Delta^i_j = \frac{(T_{\max} - {T_c}^i)}{{T_c}^i + {T^i}_{comp}}, \tag{1} $$

where ${T^i}_{comp}$ is the computation time per iteration in node i. The computation time ${T^i}_{comp}$ varies more as convergence approaches. This is because the preconditioning time required at different phases of computation depends upon the characteristics of the matrix to be preconditioned (number of non-zero elements, amount of fill-in).

6 Experiments

We present two sets of experiments to demonstrate a seamless incorporation of adaptations into pARMS using NICAN on high performance computing platform and to show that the adaptations based on the communication waiting time enhance overall performance of the application.

6.1 Adaptations for a regularly structured problem on IBM SP

The first set of experiments was conducted on the IBM SP at the Minnesota Supercomputing Institute using four WinterHawk+ nodes (375 MHz Power3 node) with 4 GB of main memory. Although each WinterHawk+ node has four processors, only one processor per node was used in the experiments. All the nodes run the AIX operating system version 4. They use a switch having peak bi-directional bandwidth of 120 MBps between each node for communication. The problem, defined in [14], is as follows: “The problem is modeled by a system of convection-diffusion partial differential equations (PDE) on rectangular regions with Dirichlet boundary conditions, discretized with a five-point centered finite-difference scheme. If the number of points in the x and y directions (respectively) are m_x and m_y, excluding the boundary points, then the mesh is mapped to a virtual $p_x \times p_y$ grid of nodes, such that a sub-rectangle of m_x/p_x points in the x direction and m_y/p_y points in the y direction is mapped to a node. Each of the sub-problems associated with these sub-rectangles is generated in parallel. This problem is solved by FGMRES(100) using a domain decomposition pARMS preconditioning. The combining phase uses Additive Schwarz procedure” [14]. To make the problem challenging for this computational environment, we consider the convection term of $2,400$ for this system of PDEs.

To trigger pARMS adaptation, the amount of the time ${T_c}^i$ spent in communications has been considered in each node. Specifically, we use the criterion that the ${T_c}^i$ normalized over the previous w outer iterations, called the window size, be nonzero. When this condition is met the number of inner iterations are increased by a constant value called dynamic addition value. The rationale behind using this criteria is that if a node shows waiting time in this iteration then it is more likely for it to show waiting time in the subsequent iteration. Also experimental observations for the waiting times for non adaptive runs showed that the waiting times are incurred in chunks with phases showing waiting times and phases showing no waiting times. Thus a rather simplistic approach of observing the previous waiting time can be used to find out the possibility of next waiting time.

Note that this criterion is local to a particular node and thus reflects only the interaction among neighboring nodes, which is acceptable due to the regular partitioning of this problem among the nodes, as shown in [14]. Otherwise NICAN may be used to exchange the global communication time data as seen in the next set of experiment. Table 1 shows results of executing pARMS with and without adaptation (column `Adapt`) on the IBM SP. In this experiment the window size w used was three. The total number of outer pARMS iterations is shown in the column `Outer Iter` followed by the total solution time `Solution`. Table 2 shows the total waiting time for each node. $R0$, $R1$, $R2$, and $R3$ indicate the ranks of the nodes. In each table, the average values over several runs are mentioned with the standard deviations shown in the brackets. Figure 4 plots the total waiting times of the nodes.

Table 1. Adaptation based on the observation of previous waiting times: On IBM SP for PDE problem

	Adapt.	Outer Iter.	Solution, s
n_0^i	yes/no		
5	no	2,000 (0.0)	1,065.0 (30.33)
	yes	713.3 (63.105)	607.89 (100.0)

Table 2. Waiting Time for nodes: On IBM SP for PDE problem

	Adapt.	Tot. Wait., s			
n_0^i	yes/no	R0	R1	R2	R3
5	no	111.5 (23.33)	50.5 (21.92)	39.0 (18.38)	75.0 (7.77)
	yes	56.3 (42.5)	38.0 (13.0)	24.66 (11.23)	44.0 (36.29)

The effect of different dynamic addition values is seen in Figure 5, left. It is seen that increasing the dynamic addition value (from 1 to 3) decreases the solution time. Figure 5, right shows that increasing the window size ($w = 10$) also decreases the solution time. From these two figures it can be concluded that either increasing the window size or increasing the dynamic addition value for a particular window size reduces the total solution time on this platform. Increasing the window size increases the amount of history information and thus helps in predicting more accurately the possibility of the next waiting time for a node. The window size determines when the adaptations are to be employed. The amount of dynamic addition determines the number of adaptations. For a given window size increasing this value balances the nodes in their inner iteration phase thus reducing the waiting time in the subsequent iterations and causing the reduction in the solution time.

A better convergence of the system of PDEs with adaptations is explained by more computations at the inner-level Krylov solver per each outer iteration. With the pARMS preconditioning the

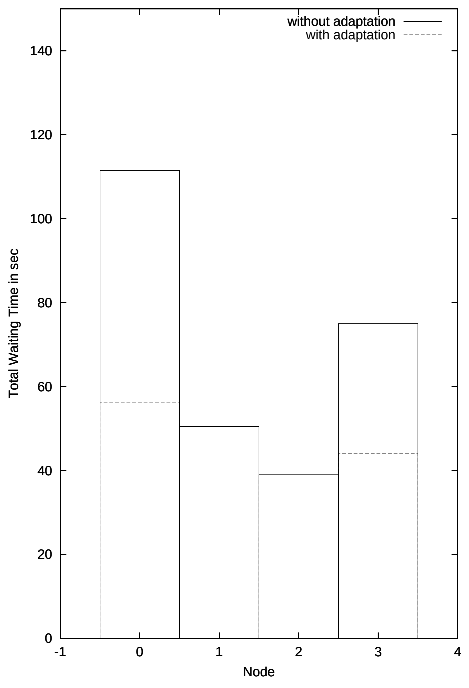


Fig. 4. Total Waiting time for nodes

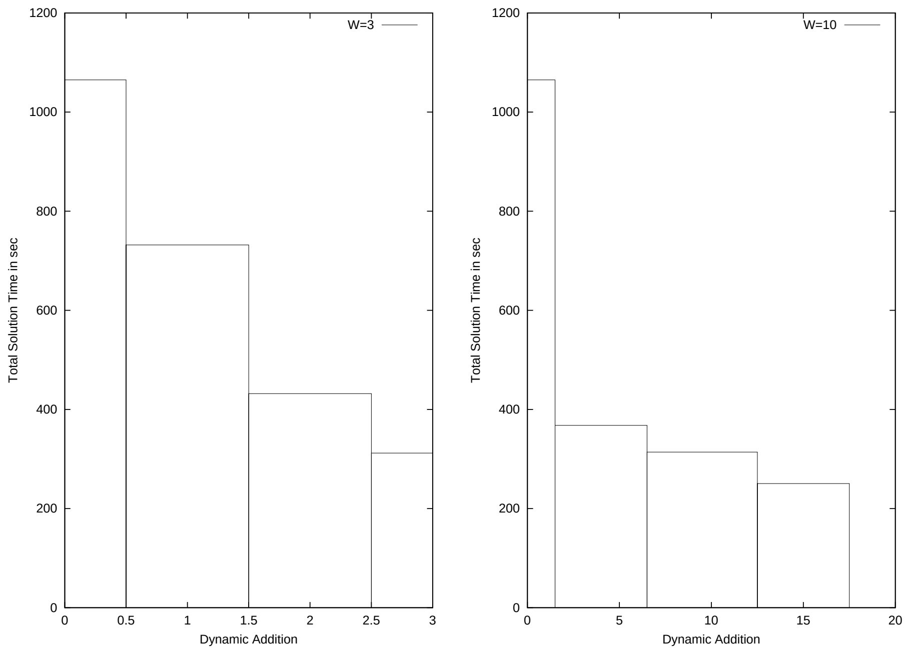


Fig. 5. Total time reduction for different dynamic additions: window sizes 3 (left) and 10 (right)

linear system is solved level by level and at the last level the system is solved using ILUT-FGMRES [9]. By using this strategy of observing the previous waiting time for a node and increasing the number of iterations for this last level solution step helps in engaging the node for more time so that it arrives at the rendezvous point later. The nodes showing waiting time are considered "faster" nodes as compared to other nodes. More local computations on nodes with "faster" communications result in the generation of more accurate local solutions per outer iteration. Upon exchange in the subsequent communication phases, this accuracy propagates to other nodes resulting in a better convergence overall. The total execution time is decreased also due to reducing of the waiting times in each outer iteration for the nodes. Consider the total waiting times in Table 2. It is seen that, with dynamic adaptation, the total waiting time has decreased in all the cases. Load balance is achieved in terms of total waiting time incurred by each node. Without adaptation the total waiting time for all the nodes show a large range. With adaptation, however, this range is narrowed.

6.2 Adaptations for irregularly structured problem on IBM SP

For the second set of experiments the problem used is taken from the field of Magnetohydrodynamic (MHD) flows. The problem as described in [5] is as follows: "The flow equations are represented as coupled Maxwell's and the Navier-Stokes equations. We solve linear systems which arise from the Maxwell equations only. In order to do this, a pre-set periodic induction field is used in Maxwell's equation. The physical region is the three-dimensional unit cube $[-1, 1]^3$ and the discretization uses a Galerkin-Least-Squares discretization. The magnetic diffusivity coefficient is $\eta = 1$. The linear system has $n = 485,597$ unknowns and $24,233,141$ nonzero entries. The gradient of the function corresponding to Lagrange multipliers should be zero at steady-state. Though the actual right-hand side was supplied, we preferred to use an artificially generated one in order to check the accuracy of the process. A random initial guess was taken" [5]. For the details on the values of the input parameters see [5].

The data transfer phase in pARMS is timed to obtain the communication time ${T_c}^i$. The formula for dynamic incrementing of the number of inner iterations shown in Section 5 equation (1) is used. Table 3 and Table 4 show that with adaptations the solution time has decreased with a corresponding decrease in the total waiting times of the nodes. Figure 6 plots the total waiting time for all the nodes. Comparing this figure with Figure 4 it can be concluded that since this problem is irregular, the overall balance in the waiting time of the nodes is not achieved. The differing time spent by each node in the preconditioning computation phase makes it difficult for the waiting time to balance for all the nodes though decrease in the waiting time for individual node is observed. Even though the problem is harder and non uniform as compared to the problem in experiments of subsection 6.1, the adaptation strategies are seen to be as useful as in the previous case.

Table 3. Adaptation based on global communication time on the IBM SP for MHD problem

n_0^i	Adapt. yes/no	Outer Iter.	Solution, s
5	no	41 (0.0)	1,021.3 (28.42)
	yes	34.5 (0.707)	781.53 (13.03)

7 Conclusion

We have proposed a framework for a distributed scientific application to learn and make use of dynamic network information through notification handlers mechanisms. The framework is imple-

Table 4. Waiting time for nodes: On the IBM SP for MHD problem

	Adapt.	Tot. Wait., s			
n_0^i	yes/no	R0	R1	R2	R3
5	no	316.0 (67.88)	570.0 (147.78)	284.0 (195.0)	15.5 (2.12)
	yes	41.5 (14.85)	413.0 (16.97)	46.0 (14.14)	16.5 (6.36)

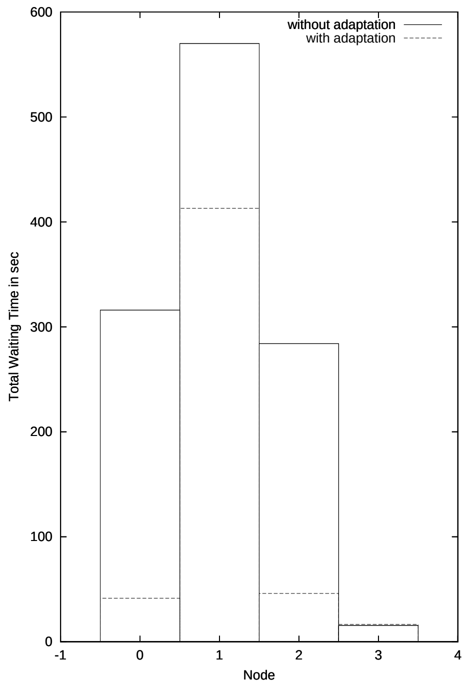


Fig. 6. Total Waiting time for: MHD problem

mented in a network information collection tool that is lightweight and requires little modifications to the user application code. The proposed framework is rather flexible since a variety of adaptation strategies can be used in a notification handler. Our network information collection tool is capable of expanding beyond existing data collecting strategies due to its modular design. The experiments show that there is an improvement in the solution time by about 43% with adaptations based on measuring previous waiting time of a node. The adaptations based on peer communication time show an improvement of about 23% as compared to non-adaptive case. Applications having pARMS like behavior can benefit from such adaptations.

Ability of NICAN to accurately capture the data transfer phase of a scientific distributed application can be enhanced by using per flow analysis. With per flow analysis NICAN will be able to correctly identify the exact start and end of the data transfer phase of the application. It would be very useful if the exact monitoring of the packets of a particular application could be done. Using such monitoring the exact time instant of these packets leaving the network interface can be found out. With more accurate throughput information it will be possible to accurately calculate the number of dynamic additions which can be employed. Thus with this approach an added amount of monitoring, along with using SNMP, can be provided to the application.

Another area in which this research can grow is that of developing a generalized framework for resource monitoring requests. An approach using eXtensible Markup Language (XML) is being tried out to allow the application to issue generalized network parameter monitoring requests. Also monitoring tools for non-IP based networks can be developed.

References

1. D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan. System support for bandwidth management and content adaptation in internet applications. In Proceedings of 4th Symposium on Operating Systems Design and Implementation San Diego, CA, October 2000. USENIX Association.:213–226, 2000.
2. J. Dongarra, G. Fagg, A. Geist, and J. A. Kohl. HARNESS: Heterogeneous adaptable reconfigurable NEtworked systems. pages 358–359, citeseer.nj.nec.com/327665.html, 1998.
3. D. Gunter, B. Tierney, B. Crowley, M. Holding, and J. Lee. Netlogger: A toolkit for distributed system performance analysis. In *Proceedings of the IEEE Mascots 2000 Conference*, 2000.
4. D. Kulkarni and M. Sosonkina. Using dynamic network information to improve the runtime performance of a distributed sparse linear system solution. Technical Report UMSI-2002-10, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2002. accepted in VECPAR 2002.
5. Z. Li, Y. Saad, and M. Sosonkina. pARMS: A parallel version of the algebraic recursive multilevel solver. Technical Report UMSI-2001-100, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2001.
6. B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A resource query interface for network-aware applications. *Cluster Computing*, 2:139–151, 1999.
7. Richard Tran Mills, Andreas Stathopoulos, and Evgenia Smirni. Algorithmic modifications to the jacobi-davidson parallel eigensolver to dynamically balance external cpu and memory load. In *Proceedings of the International Conference on Supercomputing 2001, Sorrento, Italy*, pages 454–463, June 18-22, 2001.
8. NET SNMP project. Web Site, http://net-snmp.sourceforge.net/.
9. Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS publishing, New York, 1996.
10. Y. Saad and M. Sosonkina. Distributed Schur Complement techniques for general sparse linear systems. *SIAM J. Scientific Computing*, 21(4):1337–1356, 1999.
11. Y. Saad and M. Sosonkina. Non-standard parallel solution strategies for distributed sparse linear systems. In A. Uhl P. Zinterhof, M. Vajtersic, editor, *Parallel Computation: Proc. of ACPC'99*, Lecture Notes in Computer Science, Berlin, 1999. Springer-Verlag.
12. B. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, New York, 1996.
13. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI - The complete Reference*, volume 1. The MIT Press, second edition, 1998.
14. M. Sosonkina. Runtime adaptation of an iterative linear system solution to distributed environments. In *Applied Parallel Computing, PARA'2000*, volume 1947 of *Lecture Notes in Computer Science*, pages 132–140, Berlin, 2001. Springer-Verlag.
15. M. Sosonkina and G. Chen. Design of a tool for providing network information to distributed applications. In *Parallel Computing Technologies PACT2001*, volume 2127 of *Lecture Notes in Computer Science*, pages 350–358. Springer-Verlag, 2001.
16. C. Wagner. Introduction to algebraic multigrid - course notes of an algebraic multigrid. University of Heidelberg 1998/99.
17. Richard Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.